Collaborative Editing: Improving Your Peers' Blogs

1 Overview

There is almost always room for improvement in creative writing. Sometimes it is the content and not the form that needs work, but often the form needs some fixing. There might be poorly constructed sentences, incorrect usage of words or phrases, incorrect grammatical constructions, or spelling mistakes that a spell-checker cannot catch because they are valid words (as when you type "he" instead of "the"). The content might be outstanding, but its message is weakened by the delivery. That is why people rely on human editors to read their writing and make suggestions. Blogs are no exception. Many blog posts would benefit from inspection by another pair or two of eyes.

Being able to write well is one skill; being able to read another's writing and improve it is another. A major purpose of this activity is to give you this experience, but it has other important objectives, namely:

- to give you experience collaborating with others on and contributing to open source projects,
- to give you experience reading other people's writing with the goal of improving it (i.e., constructive commenting),
- to develop your *Git* skills further, and
- to give you practice opening issues and submitting pull requests on *GitHub*.

In this activity, you will perform several roles:

- You will read the blog posts of some of your classmates and open issues on them.
- You will read the issues opened by others and try to fix them. To do this, you will fork and clone the blog post repositories, make changes in your cloned versions, push your changes back to your upstream fork, and make pull requests to the original owners with the changes that you made. Making these changes might require that you resolve merge conflicts in the process. If your suggested changes warrant discussion with the blog's owner, you will have a conversation about it through *GitHub* and ultimately find some resolution, after which the owner will close the issue.
- As the owner of a blog post repository, you might have to resolve and close issues that have been opened on your blog posts. Some issues might be trivial enough for you to fix yourself, such as format-related issues or minor punctuation or spelling mistakes, while others might have fixes proposed by one of your peers. In this case, you will receive pull requests with the suggested changes and will either accept them or have conversations in *GitHub* about them. Although the person submitting the pull request should have resolved any possible merge conflict when the request was submitted, it is possible that he or she did not, and that there is a merge conflict. In this case you should politely ask that person to resolve the merge conflict and resubmit the request.

2 Prerequisite Knowledge

Before you work on this project you should have read the chapters in the *Pro Git* book named *Git Basics* and *Git Branching*. You need to understand elementary concepts about *Git* repositories, branches, and working with branches and remote repositories. You should know how to use the most common *Git* commands on the command line. You should know basic *Markdown* in case you need to suggest changes in the *Markdown* used to format blog posts. You should also be familiar with the *GitHub* interface.

3 Detailed Instructions

These instructions are divided into three categories corresponding to the three different roles that you might perform.

3.1 Opening An Issue

Suppose that you have been asked to review the blog post of a user named `Shakespeare`.

1. Your goal is to open issues on *GitHub* for one or more of `Shakespeare`'s posts. Examine the post first for formatting problems. To do this you must be reading the web-page version of the blog, i.e., the one whose URL ends, for example, in "`github.io/shakespeare-weekly`". Read each post first for spelling errors, then for grammar errors that you can detect, then usage. Grammar errors are sometimes obvious. For example, "*who argued so* ***good***" should be "*who argued so* ***well***." Usage is when the wrong words or expressions are used in a sentence, like "*Quicksort was faster* ***then*** *selection sort*" should be "*Quicksort was faster* ***than*** *selection sort.*" For each mistake or weakness that you find, open a separate issue. If you think a sentence is just bad for one reason or another, open an issue and try to state constructively how it could be improved.

2. To open an issue, click the `Issues` tab near the top of the page, and on that tab, click the green button on the right hand side labeled "`New Issue`". In the page that opens, fill in the `Title` text box with a good and short title, and then describe the issue in the area below that. You should use links to the posts as needed, or copy and paste the text in question. When you are finished, click the green `Submit Issue` button below the text area.

3.2 Resolving Issues and Submitting Pull Requests

Caveat: The workflow that is described here is not a rigorous workflow that you would use if you were trying to contribute to an open source project with people whom you do not know personally. It is an easier workflow to follow, chosen because you might be asked to do this exercise before you learned some of the more advanced features of *Git* that you would need to know to follow a more rigorous workflow.

Suppose that you have been browsing the blog posts of various people and that you have found an open issue in one of `Shakespeare`'s posts that you think you can fix.

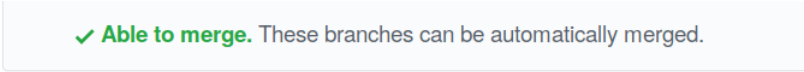
The workflow that you must use is as follows:

1. Fork a copy of `Shakespeare`'s repository.
2. Clone this fork to your local machine.
3. Make whatever changes you think are necessary to the blog post in your cloned copy.
4. When you think that a particular change is good and complete, stage it using the `git add` command, and commit the staged change with a good commit message, using the `git commit` command.
5. Repeat the preceding instruction for each independent change that you make. If there are multiple mistakes that you are trying to fix, or improvements that you are trying to make, each should have its own commit with a distinct and descriptive commit message. This way it is easier to keep track of what was done, and to accept or reject them independently.
6. When you are all finished with these changes, you are ready to push them to your upstream fork of `Shakespeare`'s repository. [1]
7. Push your changes to your remote using the `git push` command.

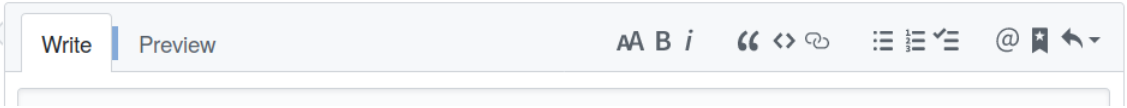
[1] This is the step that is simplified. It omits a synchronizing step: Before you push your changes to your upstream fork, you should really pull down the most recent copy of `Shakespeare`'s blog, in case there were changes made to it since you forked it, and merge those changes locally. If you know how to do this, you can do it now, otherwise proceed without this step.

8. You are now ready to issue a pull request. Open the *GitHub* page with *your fork* of `Shakespeare`'s repository. Click on the gray button labeled "`New Pull Request`". *GitHub* will display a page with the heading "`Comparing changes`".

   (a) If you are able to merge your changes into `Shakespeare`'s repository, you will see some green text stating, "`Able to merge`" :

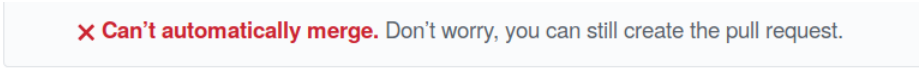
   

   and there will also be a button labeled "`Create new pull request`".
   Click on it and fill in the upper text box with a brief message and the larger one below it with a very detailed message. The message should reference the issue that the pull request resolves, and should be addressed to the *GitHub* user who "owns" the repository. The edit area has a toolbar that you can use to do this:

   The `@` symbol precedes the *GitHub* username to whom you want to address the message. A `#` followed by a number references an issue number. For more information about formatting and editing in the GitHub interface, see Writing on GitHub.
   Then click the "`Create pull request`" button. You are finished with the submission of the pull request. (This does not mean it will be accepted without further work on your part.)

   (b) If *GitHub* is not able to merge your changes automatically, you will see instead the message

   Click the "`View Pull Request`" button, and in the new page, click the gray "`Resolve conflicts`" button. *GitHub* will now display the portion of the file with the two different pieces of text that must be resolved. It will look something like this:

```
<<<<<<< master
You can submit *Pull Requests* from your fork to the original repository
to help make other people's projects better by offering your changes up
to the original project.
=======
You can submit *Pull Requests* to the original owner to help make other
people's projects better by offering your changes up to the original project.
>>>>>>> master
```

   You need to edit the paragraphs in place, deciding how to combine them, and deleting the lines with the "<", "=", and ">" symbols. Then click the "`Mark as resolved`" button and then the "`Merge commit`" button that is displayed. You are finished with the submission of the pull request. (This does not mean it will be accepted without further work on your part.)

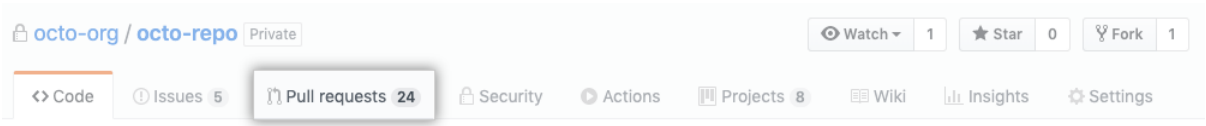
3.3 Closing Issues

If there is an issue on your repository that is very trivial, you can make the changes and close it yourself. It is fairly simple to do this; just use the editor in *GitHub* on the file that needs the changes, return to the `Issues` list, select the issue, write a very descriptive comment, including an acknowledgment and a word of thanks to the person who opened the issue and/or the one who submitted the fix for it, and click the "`Close Issue`" button on the page that displays. Finally, you should go back to your local copy of the repository and pull down the latest changes using the `git pull` command.

3.4 Accepting and Closing Pull Requests

In this case you are the owner of the blog and you need to review submitted pull requests and decide what to do with them. The instructions for doing this are not the ones you would normally use for working in an open source project as a contributor because they are simplified for this exercise. Normally you would pull down your repository and create separate branches locally to make the changes, and then push them back up, but for this exercise you will work directly in *GitHub*. Again, the reason for this is that in this exercise there is no assumption that you have mastered enough *Git* to perform the more advanced workflow.

1. When you, as the owner of a *GitHub* repository, receive a pull request from someone, you should review the request and then decide if it is acceptable. If so, you need to merge the changes into your repository. If not, you should have a conversation with its author. On your repository page, click on the "`Pull Requests`" tab:



2. In the "`Pull Requests`" list, click the pull request you'd like to review and close.
3. Click on the "`Files Changed`" link near the top of the page to review the differences between your current file and the proposed changes in the request.
4. Click on the `Review Changes` button. Decide which option you want to choose and then click `Submit Review`. If there is a merge conflict, you should request the person submitting the request to resolve the conflict and resubmit.
5. Assuming that there are no merge conflicts, you should see a green "`Merge Pull Request`" button. Click this button to merge the changes into your current branch. You have finished the task.